أساسيات وتطبيقات

لغـــة ســـــي

C

د. عمر محمد زرتي
أستاذ بقسم الحاسوب
كلية العلوم، جامعة الفاتح، طرابلس

بسم الله الرحمن الرحيم

إنما يخشى الله من عباده العلماء

صدق الله العظيم

تمهيد

الحاسوب آلة عجيبة..

فهو آلة تقوم بالعديد من المهام المختلفة.

عن طريقه يمكنك حل المسائل الرياضية، وطباعة الرسائل والتقارير، والاتصال الهاتفي، والبريد الاكتروني، كما يمكنك قراءة الأخبار، والحصول على المعلومات في شتى المجالات. وإذا كنت مهندسا معماريا يمكنك تصميم العمارات باستخدام الحاسوب، كما يمكن للطيارين التدريب على الطيران بطريقة ما يعرف بالمحاكاة باستخدام الحاسوب...

من كل هذه الاستخدامات ، وغيرها كثير، يصبح واضحا مدى أهمية تعلم الحاسوب وما يتعلق به من علوم..

وعلوم الحاسوب كثيرة، ومجالها واسع.. ولعل أهمها ما يعرف ببرمجة الحاسوب..

والبرمجة تعني كتابة مجموعة من الأوامر، بإحدى اللغات المعدة لهذا الغرض، يقوم الحاسوب بتنفيذها لتحقيق هذف معين.

وهذا الكتاب { أساسيات وتطبيقات لغة سي } يتناول إحدى أهم لغات البرمجة، بأسلوب مبسط، يستهدف طلبة المراحل الأولى من التعليم العالي، ليكون مرجعا لهم يساعدهم في تعلم قواعد هذه اللغة وتطبيقاتها.

آمل أن يجد أبنائي الطلاب في هذا الكتاب الغرض المطلوب، ، وأن يجد فيه زملائي من هيئة التدريس مساعدا لهم في تدريس لغة سي.

وأخيرا لايفوتني أن أقدم شكري وتقديري لكل من ساهم في إعداد هذا الكتاب سواء بالمراجعة العلمية أو اللغوية أو الإخراج الفني والطباعة. وأخص بالذكر الأخ الأستاذ عمران محمد أبوميس على ملاحظاته اللغوية القيمة، ولا أنسى ابني المهندس أنس زرتي الذي قام بعمليات التجميع والإخراج باستخدام الحاسوب.

والله نسأل السداد والتوفيق للجميع لما فيه الخير والصلاح..

د. عمر زرتي

الفهرس

الباب الأول : مقدمة

الباب الثاني : الثوابت والمتغيرات وجُمل التعيين

الباب الثالث: الإدخال والإخراج Input & Output

الباب الرابع: الجمل الشرطية Conditional Statements

الباب الخامــس: الحلقات Loops

الباب السادس : المصفوفات والنضائد Arrays & Strings

الباب التاسع : الدوال Functions

الباب العاشر :البيانات المركبة Structured Data

الباب الحـــادي عشر : ملفات البيانات Data Files

الباب الثاني عشر : تطبيقات Applications

الملاحق

1 الباب الأول

مقدمـــة

Introduction

1.1 لغــات البرمجة

إن أي لغة ما هي إلا مجموعة من الكلمات والقواعد ، حيث يوجد لكل كلمة معنى معين ، وباستخدام القواعد يمكن تكوين جمل مفــــــيدة مـــن هـــذه الكلمات . فإذا تفاهم مجموعة من الناس على لغة معينة ، أصبح بمقدورهم نقل الأفكار من واحد إلى آخر ، وبذلك يتم التعاون بينهم من أجل حياة افضل .

وعندما اخترع الإنسان " الحاسوب " لغرض تسخيره فـــي معالجـــة البيانـــات وتخزينها وإجراء العمليات الحسابية عليها، كانت المشكلة التي واجهت العلمـــاء هي تحديد وسيلة تخاطب مع هذه الآلة التي تختلف عن الإنسان فـــي تركيبهـــا وقدراتها . فهذه الآلة لا يمكنها أن تميز إلا حالتين فقط ، يمكن أن نرمز لهمـــا بالرمزين {0،1} ، حيث قد يرمز الصفر لدائرة إلكترونية مفتوحة (أي لا يمر فيها تيار) ويرمز الواحد لدائرة إلكترونية مغلقة (أي يمر فيها تيار).

إلا أن تحديد استخدام الصفر والواحد فقط في جميع الكلمات الموجهة للآلة (وهو ما يعرف بلغة الآلة machine language) لم يكن مناسبا إطلاقـــا للإنـــسان المبرمج بهذه اللغة . فقد كان الوقوع في الخطأ مثل استبدال الـــصفر والواحـــد بدون قصد أمرا شائعا ، ناهيك عن صعوبة قراءة برنامج يتكون من سلسلة من الأصفار والواحد .

من هنا جاءت فكرة (البرنامج المترجم) حيث تم الاتفاق على استخدام بعـــض المصطلحات الواضحة مثل (ADD المجمـــع ، MUL للضرب) في كتابة

البرنامج، وحيث أن هذه المصطلحات لا يفهمها الحاسوب ، تم إعـداد برنـامج يسمى المجمّع assembler الذي يترجم هذه المصطلحات إلـى لغـة الآلـة ، وسميت مجموعة هذه المصطلحات بلغة التجميـع assembly language .
بـهذا تكون مشكلة إعداد برامج واضحة في مصطلحاتـها قد تم حلها إلى حد بعيد ، ولكن بقيت مشكلة أخرى ، وهى التصاق هذه اللغة بالمعالج الذي صممت من أجله. أي أن البرنامج المكتوب لمعالج معين ، لم يكن صـالحا أو مفهومـا لمعالج آخر.

لذلك تم تصميم وإعداد لغات توصف بأنها عالية المستوى high-level، حيـث تتمتع هذه اللغات بوضوح أكثر في التراكيب اللغوية ، أي أنها اكثر قربا مـن لغات الإنسان منها إلى لغة الآلة .

كما تتمتع هذه اللغات عالية المستوى بخاصية الاسـتقلالية ، أي أن البرنـامج المكتوب بلغة عالية المـستوى مثـل FORTRAN أو PASCAL أو C يعتبر صالحاً ومفهوما من قبل مختلف معالجات الحاسوب طالما توفر البرنامج المترجم (ويسمى إما مصرف Compiler أو مفسر Interpreter) بالجهـاز المنفذ لهذا البرنامج .

وتختلف اللغات عالية المستوى في تركيبها باختلاف الغرض الذي صممت مـن أجله . فبينما تهدف لغة فورتران FORTRAN إلى توفير وسـيلة برمجيـة للمهندسين والفيزيائيين وغيرهم من ذوى التخصصات العلمية ،نجد أن لغة مثل COBOL مصممة من أجل التطبيقات الإدارية والتجارية .

وتعتبر لغة سى C موضـــوع هذا الكتاب من اللغات عالية المستوى ، وهي تجمع بين مزايا اللغات العلمية والإدارية واللغات المستخدمة في برمجة الأنظمة الخاصة بتشغيل الحاسوب.

تعتبر لغة سي من أكثر لغات البرمجة استعمالا في التطبيقات العلمية والتجارية وأيضا في برمجة نظم التشغيل . ويرجع الفضل في إنشاء هذه اللغة إلى دنـــس رتشي Dennes Richie المبرمج بشركة بل Bell المعروفة اليــوم بـــشركة AT&T Bell . وقد ظهر أول وصف دقيق لهذه اللغة سنة 1978 علـــى يـــد براين كرنيهان Brian Kernigham مع مؤلف اللغة نفسه (دنس رتشي) في كتاب من تأليفهما.

يهدف هذا الكتاب (أساسيات وتطبيقات لغة سي) إلى أن يكون مرشدا ومساعدا يساهم في تبسيط عملية تعلم قواعد وتطبيقات لغة سي C ، حسب المنهج المقرر في أغلب الجامعات والمعاهد العليا، مستخدما في ذلك الصيغة الشائعة الاستعمال TURBO C (وتعني سي السريعة) ، وذلك لما تتميز به هذه الـــصيغة مـــن بساطة في إعداد البرنامج نظرا لشمولها على ما يعرف بمحرر النصوص text editor . أضف إلى ذلك بساطة التنفيذ ، وسرعته ، ومـــساعدة المـــتعلم فـــي اكتشاف الأخطاء اللغوية في برنامجه .

1.2 لغة سي (صيغة TURBO C)

نتعرض هنا إلى أهم الوظـائـف التي يـحتاج إلـيها المتدرب لـدى استخدامه Turbo C. ولمزيد من الاطلاع على الإمكانيات التي لم يرد ذكرهـا في هذه العجالة يجب الرجوع إلى الدليل الأصلي لهذه المنظومة ، علمـا بـأن مفتاح المساعدة F1 يقدم الكثير من المعلومات عن وظائف العديد من المفـاتيح في تشغيل هذه المنظومة، وكذلك عن التراكيب اللغوية وقواعد لغة سي .

ولتحقـيق هذه الأهداف ، يعرض علينا تربو سي turbo C قائمـة رئيـسية main menu في شريط أفقي، وهي تشمل مجموعة من الاختيارات أهمها :

1. الأمر File، وهو يمكننا من الحصول على قائمة عمودية من الاختيارات تشمل الآتي :
   - فتح ملف جديد (أي برنامج جديد) بواسطة الاختيار New.
   - تخزين الملف بواسطة الاختيار Save.
   - فتح ملف قديم بواسطة الاختيار Load.
   - تخزين ملف تحت اسم جديد بواسطة الإختيار Save as.
   - إنهاء العمل بالاختيار Quit.
2. الاختيار Edit لتعديل الملف من حيث الإضافة insert والإلغاء delete.
3. الاختيار Compile لترجمة البرنامج واكتشاف الأخطاء وتصحيحها.
4. الاختيار Run لتنفيذ البرنامج.

لاحظ أن الضغط على الزر F10 يمكّنك دائما من الدخول على القائمة الرئيسية، وأن زر السهم الأيمن → و زر السهم الأيسر ← يمكناك من الانتقال من اختيار إلى آخر في الشريط الأفقي. اضغط على الزر Enter لتنفيذ ذلك الاختيار لتحصل على قائمة عمودية للاختيارات التابعة له. عند ذلك يمكنك استعمال زر السهم الأسفل ↓ أو زر السهم الأعلى ↑ للتحول إلى أي اختيار في هذه القائمة العمودية.

لا تنس أن المفتاح F1 موجود دائما لمساعدتك في الحصول على أي معلومة تخص التشغيل أو اللغة ، ولكن ذلك يتطلب منك إجادة اللغة الإنجليزية للاستفادة من هذه المساعدة.

1.3 مراحل إعداد البرنامـــــج وتنفيذه

نـــلاحظ أولا أن الحـــصول على نتـــائج من أي برنامج يتطلب الآتي:

1. مرحلة إعداد البرنامج المكتوب بلغة عاليـــة المـــستوى، ويـــسمى هـــذا البرنامج بالمصدر source . ويتم تخزين هذا البرنامج إما على القريص diskette أو على القرص الصلب hard disk.

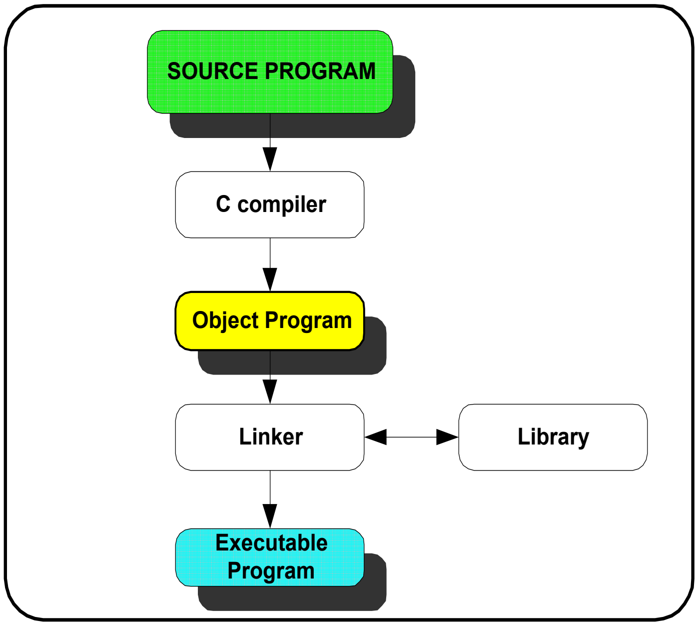
2. مرحلة ترجمة البرنامج compilation وفيها يتم تحويل البرنامج إلـــى لغة الآلة بعد التأكد من خلوه من الأخطاء اللغوية حسب قواعـــد اللغـــة

المستخدمة . وإذا وجدت أي أخطاء يطلب المترجم تصحيحها، ولا يقوم بعملية الترجمة إلا بعد الانتهاء من تصحيح كل الأخطاء.

3. مرحلة التنفيذ execution ويتم فيها إدخال البيانات المطلوبة ، ثم يتم تنفيذ تعليمات البرنامج .

مترجم لغة سي لا يقوم بترجمة البرنامج المصدر source program إلى برنامج مكتوب بلغة الآلة (ويسمى بالبرنامج المستهدف object program) إلا بعد التأكد من خلو المصدر من أي أخطاء لغوية compiler error . وقبل تنفيذ البرنامج المستهدف يجب أن يتزود البرنامج ببعض الدوال التي يحتاجها أثناء التنفيذ ، وهذه الدوال توجد فيما يسمى بالمكتبة library ، وهي عبارة عن ملف يحتوي على تعاريف لهذه الدوال . لذلك نحتاج إلى برنامج يسمى الرابط linker يقوم بربط البرنامج المستهدف بهذه المكتبة (كما مبين بالشكل 1.3.1) . أي أن الرابط يقوم بعملية البحث في المكتبة عن أي دالة تم استدعاؤها في البرنامج. فإذا وجدها يتم تنفيذ البرنامج ، وإذا لم يجدها فإنه يصدر رسالة خطأ error message ويسمى الخطأ في هذه الحالة من نوع خطأ الربط linker error . أي أن هناك نوعين من الأخطاء قد يظهرا قبل تنفيذ البرنامج وهما :

1. **الخطأ اللغوي** **compiler error** : يتم تصحيحه بالرجوع إلى نقطة الخطأ في البرنامج ، فإذا كان الخطأ واضحا كما هو الحال في أغلب



يتم تصحيحه ،وإعادة الترجمة من جديد. أما إذا عجز المبرمج عن اكتشاف هذا الخطأ فعليه بالرجوع إلى دليل اللغة أو الاستعانة بمفتاح المساعدة F1.

2. **خطأ الرابط** **linker error** : يتم تصحيحه بالتأكد من كتابة اسم الدالة

لاحظ أن تصحيح الأخطاء اللغوية في البرنامج لا يعني أن البرنامج سوف يتم تنفيذه بصورة سليمة فقد تظهر فيه أخطاء أثناء التنفيذ تسمى بأخطاء التنفيذ execution errors مثل خطأ الفيض overflow الذي ينتج عند محاولة القسمة على الصفر في البرنامج.

بصورة صحيحة، أو التأكد من استخدام متغيراتها كما هو محدد في تعريفها.

وأخيرا يجب التنويه إلى ما يعرف بالأخطاء المنطقية التي لا علاقة لها بلغة البرنامج أو الآلة التي تنفذه. بل هى أخطاء ناتجة عن بنـــاء البرنـــامج علــى خوارزمية غير سليمة . ولتصحيحها لابد من مراجعة هذه الخوارزمية خطــوة خطوة بطريقة التتبع.

الباب
الثاني
2

الثوابت و المتغيرات وجمل التعيين

Constants, Variables and Assignment Statements

2.1 مقـدّمــة

يتكون البرنامج في لغة سي من دالة رئيسية واحدة اسمها main، مع إضافة بعض الدوال الأخرى (إلى جانب هذه الدالة) إذا تطلب الأمر.

والدالة هي عن مجموعة من الجمل تحت اسم معين . وتوضع هذه الجمل بين القوسين { } بعد اسم الدالة متبوعا بالقوسين ( ) .

فمثلا قد تكون الدالة الرئيسية كما في الشكل (2.1.1) .

```
main (   )
{ printf ( " HALLO " ) ;  }
```

شكل (2.1.1)

في هذه الدالة نجد جملة واحدة فقط هي :

```
printf ( " HALLO " ) ;
```

وهى عبارة عن أمر طباعة لكلمة (HALLO) . لاحظ أن الجملة في لغة سي تنتهي دائما بالفاصلة المنقوطة (;) .

إذا استخدمنا المترجم turbo C في تنفيذ هذا البرنامج البسيط سوف نتحصل على الناتج المذكور ، وهو طباعة كلمة (HALLO) ، ولكن بعض المترجمات الأخرى قد تتطلب وضع التوجيه

```
# include  < stdio.h >
```

قبل الدالة main وذلك لوجود جملة طباعة بالبرنامج ، و سنناقش الغرض من هذا التوجيه directive فيما بعد .

لاحظ أن هذا البرنامج غير عملي، لكنه يمثل برنامجا صحيحا بلغة C يؤدى غرضا معينا.

فالدالة، سواء أكانت الرئيسية main أم غيرها، عادة ما تحتوى على مجموعة من الجمل المختلفة . لاحظ أن الجملة في لغات البرمجة تختلف عن الجملة في اللغات العادية التي يستخدمها البشر في التخاطب بينهم، فالجملة في لغات البرمجة هي عبارة عن أمر من المبرمج للحاسوب بأن ينفذ عملا ما.

فمثلا الجملة:

```
x = 5;
```

هي طلب تعيين القيمة 5 للمتغير x، لذلك فهي تسمى جملة تعيين Assignment statement . والجملة :

```
sum = 27.8 + 33.5  ;
```

تطلب من الحاسوب أن يقوم بعملية الجمع 33.5 + 27.8 وتعيين الناتج (أي 61.3) للمتغير sum . ورغم أن هذه الجملة تطلب أمرين هما الجمع والتعيين، إلا أنها تسمى جملة تعيين . أي أن جملة التعيين تأخذ الشكل العام :

```
var = expression ;
```

حيث var يرمز لأي متغير ، و expression رمز لأي عبارة حسابية .
سندرس فيما بعد تركيبة العبارات الحسابية ، ولكن يمكننا الإشارة الآن إلى أن :

المتغير : هو اسم يرمز لموقع في الذاكرة ، ويتكون من مجموعة من الأحرف اللاتينية:

a , b , c , ... z
A , B , C , ... Z

والأرقام :

0 , 1 , 2 , 3 , ... 9

بشرط أن يبدأ (من اليسار) بحرف (وليس برقم) . كما يمكن استخدام الشرطة السفلية (_) underscore في الاسم، وهى تستخدم عادة لربط كلمتين لتكوين اسم واحد . لاحظ ضرورة التمييز بين الشرطة السفلية (_) وإشارة الناقص (-) فهذه الأخيرة غير مسموح بها في اسم المتغير.

مثال : الأسماء التالية جائزة كأسماء متغيرات :

student_name
M323

income
computer
sum_of_x

لاحظ استخدام الأحرف الصغيرة في أسماء المتغيرات بـدلا مـن الأحـرف الكبيرة ، وذلك تـميـيزا لأسماء المتغيرات عن أسماء الثوابت .وهذه القاعدة ليست ملزمة ولكنها عادة تم الاتفاق عليها في كتابة برامج لغة سى ، وسـوف نقوم باتباعها . لاحظ أيضا أن لغة سي تميز بين الحرف الصغير والكبير، فمثلا المتغير cBook لا يكافئ المتغير cbook في لغة سي.

<u>مثال</u> : الأسماء التالية قد تستخدم كثوابت في برنامج بلغة سي:

PI
ENTER
MAX_X
NUMBER_OF_STUDENTS

والمثال التالي يبين بعض الأخطاء الشائعة في تسمية المتغيرات.

<u>مثال</u> : الأسماء التالية غير جائزة في لغة سى .

الاسم	*الخطـأ*

item-price	لا يجوز استخدام إشارة الطرح في الاسم
4M	يجب البداية بحرف من اليسار وليس برقم.
Why?	لا يجوز استخدام الإشارة ? في الاسم
int	كلمة محجوزة لتعريف الأعداد الصحيحة

اسم المتغير في لغة سى ليس له علاقة بنوعه (مثلما هو الحال في لغة فورتران مثلا)، لان المترجم C لا يفترض نوعا معينا من عنده بل يجب على المبرمج أن يحدد من بداية الدالة أنواع جميع المتغيرات التي يستخدمها في البرنامج . ويجوز تعريف الأنواع خارج الدالة أو داخلها، ولكن لكل وضع معنى خاص به سنتعرض إليه فيما بعد .

والمقصود بتعريف نوع المتغير هو تحديد ما إذا كان مثلا من نوع الصحيح integer أو الكسري float أو الرمزي char ... الخ . والسبب في تحديد النوع من البداية هو ضرورة معرفة الحيز الذي يحتاجه هذا المتغير في ذاكرة الحاسوب وطريقة تمثيل القيم التي ستوضع له في ذلك الحيز .

2.2 النوع الصحيح int

تستخدم الأعداد الصحيحة في عمليات العد . فكما هو معلوم هناك أشياء قابلة للعد مثل عدد الطلبة في الكلية ، وعدد السكان، وعدد أجهزة الحاسوب في الجامعة ، ... الخ. وهى أعداد صحيحة أي لا يوجد بها كسور، ويخصص لها في ذاكرة الحاسوب حيز (يسمى الكلمة word) يتكون من 16 بت (خانة ثنائية)، وإذا كان هذا الحيز غير كاف، هناك إمكانية في لغة سي لتغييره إلى نوع 32 بت. و لمعرفة الحيز بالبايت للمتغير الصحيح يمكنك استخدام الدالة sizeof .

لاحظ أن الأعداد الصحيحة قد تكون موجبة أو سالبة ، لذلك فإننا نحتاج إلي تخصيص خانة واحدة على الأقل لإشارة العدد، ويبقى باقي الحيز لقيمة العدد.

لتحديد أن المتغير k من النوع الصحيح (سواء أكان سالبا أم موجبا) نكتب

```
int   k ;
```

في بداية الدالة الموجود بها هذا المتغير .

وإذا كانت هذه الدالة تحتوي على أكثر من متغير صحيح يمكن تعريف المتغيرات في جملة واحدة ، مثل :

```
int   count , sum , total   ;
```

حيث يتم تعريف المتغيرات (count , sum , total) بأنها من النوع الصحيح. ويجوز أيضا دمج التعيين وتحديد النوع في جملة واحدة مثل :

```
int  m = 5 ;
```

لعرض قيمة المتغير الصحيح على الشاشة ، نستخدم الدالة printf على النحو:

```
printf ( " % d " , m ) ;
```

حيث m ترمز هنا للمتغير الصحيح المطلوب طباعته على الشاشة ، أما " %d " فهي نضيد يصف للمصرّف C كيف يتم عملية الطباعة ، ولذلك فإن هذا النضيد يسمى عادة بالهيئة (أو الشكل format) . أما حرف d في هذا النضيد فيرمز إلى أن العدد المطبوع هو من النوع الصحيح العشري decimal .

مثال <u>(2.2.1)</u> : كم عدد المتغيرات في البرنامج التالي وماذا يطبع عند تنفيذه ؟

```
main ()
{
        int k, m , sum ;
        k = 23;
        m = 45;
        sum = k + m;
        printf("%d",sum);
}
```

الشكل (2.2.1)

متغيرات هذا البرنامج هي:

k , m , sum

أي أن البرنامج توجد به 3 متغيرات وهي جميعها من النوع الصحيح int بناء على التعريف الوارد في أول جملة من جمل الدالة main .
الناتج المطبوع على الشاشة سيكون 68 وهو ناتج عملية الجمع 23+45 ، ولكن قد لا يبدو لك ذلك واضحا بل قد تجد أن الناتج غير ذلك . والسبب في ذلك أننا لم نطلب في البرنامج طباعة الناتج في سطر جديد . فمثلا إذا كان السطر الحالي على الشاشة به الرقم 2 ، فستقرأ الناتج على أنه 268 . لذلك نقوم بتعديل جملة الطباعة السابقة حتى يطبع الناتج في أول سطر جديد ، على النحو التالي :

```
printf ( " \ n % d " , sum );
```

حيث أضفنا (n \ ) في هيئة الطباعة وذلك يعني البداية في سطر جديد .

لاحظ أن استخدام الحرف x في هيئة الطباعة بدلا من الحرف d سيعطي الناتج بالنظام السادس عشري hexadecimal وليس بالنظام العشري decimal.

مثال <u>(2.2.2)</u> : ماذا يطبع البرنامج التالي؟

```
main ()
{
   int dec=256;
  printf("\n%x",dec);
}
```

الشكل (2.2.2)

هذا البرنامج يطبع العدد 100 ؟ من أين جاء هذا العدد ؟ فنحن عيّـنّا للمتغيـر dec العدد 256 وليس 100. السبب يكمن في استخدام الحرف x فـي هيئـة الطباعة بدلا من d ، والحرف x يستخدم – كما ذكرنا– لطباعة الأعداد بالنظام السادس عشري ، فكما هو معلوم فإن 100 في النظام السادس عشري تتـاظر 256 في النظام العـشري لأن:

$$(100)_{16} = 1 \times 16^2 = (256)_{10}$$

كما يمكنك طباعة أي عدد عشري بالنظام الثماني إذا أردت ذلك. فمثلا يمكنك طباعة العدد 256 بالنظام الثماني وذلك باستخدام الحرف o بدلا مـن x في البرنامج بالشكل (2.2.2) لتجد أن الناتج هو $(400)_8$ لأن :

$$(400)_8 = 4 \times 8^2 = 4 \times 64 = (256)_{10}$$

دعنا أخيرا ، مدفوعين بحب الاستطلاع ، نطبع الحيز المخصص للنوع الصحيح في المصرّف turbo C وذلك بالاستفادة من الدالة sizeof على النحو التالي :-

```
main (   )
{   int  i  ;
    printf ( " \ n % d " , sizeof (i) )  ;
}
```

الشكل (2.2.3)

عند تنفيذ هذا البرنامج سيطبع على الشاشة الرقم 2 ، دليلا على أن طول الحيز المخصص للعدد الصحيح هو 2 بايت أي 16 بت .
وإذا خصصنا خـــانة للإشارة ، فإن 15 بت فقط تبقى للقيمة المطلقة ، وهذا ينبّــهنا إلى أن أكبر عدد صحيح (من نوع int) يمكن تخزينه هو :

$$(0111........111)_2 = 2^0 + 2^1 + 2^2 + \ldots + 2^{14} = 2^{15} - 1 = 32767$$

وإذا زادت القيمة عن هذا الحد سيحدث خطأ (يعرف بخطأ الفيض over-flow) وينتج عنه حسابات غير صحيحة .

2.3 النوع العائم float

يستخدم النوع العائم لتمثيل الأعداد الكسرية بطريقة النقطة العائمة، حيث تجــزأ الكلمة (word) المستخدمة لتخزين العدد إلى جزءين ، الجزء الأول لتخــزين الكسر ، والثاني لتخزين الأس .
فمثلا العدد الثنائي $(1101.01)_2$ يمكن كتابته على الصورة :

$$(1101.01)_2 = (0.110101)_2 \times 2^3$$

حيث :

الكسر = .110101

الأس = $(3)_{10} = (11)_2$

من الواضح هنا أن الأس يحتاج عادة إلى حيز أقل من الكسر، لذلك فإن الحيــز الكلي للكلمة (وهو غالبا ما يكون 4 بايت = 32 بت) يقسم بحيث ينال الجزء الكسري عددا أكبر من الخانات . فمثلا تقسم الكلمة ذات 4 بايــت إلى 25 خانة للجزء الكسري (بما في ذلك الإشارة)، و 7 خانات للأس (بما في ذلك إشارة الأس) .

في لغة سى يتم تعريف المتغير بأنه من النوع العائم باستخدام float كما فــي البرنامج التالي :

```
main (   )
{
   float  t1 , t2  , sum   ;
   t1 = 35.56  ;
   t2 = 26.357  ;
   sum = t1 + t2  ;
   printf ( " \ n % f " , sum )  ;
}
```

الشكل (2.3.1)

نلاحظ في هذا البرنامج ما يلى :

1 . تمّ تحديد أن المتغيرات t1 , t2 , sum من النوع العائم في جملة واحدة هي:

float t1 , t2 , sum ;

2 . تمّ تعيين قيم كسرية للمتغيرين t1 , t2 ومجموعهما للمتغير sum . لاحظ أنه كان من الممكن تخصيص القيم أثناء تحديد النوع ، على النحو:

float t1 = 35.56 , t2 = 26.357 , sum ;

3 . تمّ استخدام الحرف f في هيئة الطباعة

printf (" \ n %f " , sum) ;

حيث يبين الحرف f أن المطلوب طباعته هو من النوع float. بهذه الطريقة يطبع الناتج على الصورة 61.917000 حيث يوجد 6 خانات عشرية

كسرية بالناتج . سنوضح فيما بعد طرق التحكم في عدد الخانات المطبوعة من الجزء الكسري ، ولكن هنا سنكتفي بهذه الطريقة البسيطة لطباعة الأعداد الكسرية .

إذا أردنا طباعة الأعداد على صورة النقطة العائمة :

$$a \times 10^n$$

يمكننا استخدام الحرف e في هيئة الطباعة بدلا من f على النحو:

```
printf ("\n  %e",sum) ;
```

فإذا كان sum = 61.917 فإن ناتج تنفيذ هذه الجملة يكون :

6.19170e+01

حيث يفصل الحرف e بين الكسر a والأس n . وإذا كان

wt = 0.00123

فإن ناتج تنفيذ الجملة:

```
printf ("\ n  %e" , wt) ;
```

سيكون

1.23000e-03

حيث نلاحظ أن الأس سالب لأن العدد wt أقل من واحد . أي أن الناتج يكـــون بصورة عامة على الصورة :

$$n \pm a\ e$$

حيث n عدد صحيح و:

$$1 \leq a < 10$$

و يمكن تعيين قيمة لمتغير عائم على النحو:

$$x = 5.23\ e+3$$

وهى تكافئ x = 5230.0 .

مثال (2.3.2) : ماذا يطبع البرنامج التالي عند تنفيذه؟

```
main ( )
{
        float   real_value = 1234.56  ;
        int   word_size   ;
        word_ size = sizeof ( real_value );
        printf ( " \ n % f    % d " ,
               real_value , word_ size);
}
```

الشكـــل (2.3.2)

ناتج تنفيذ هذا البرنامج هو

1234.560059 4

حيث العدد 4 هو حيز الكلمة word size المخصص للنوع float ، أما العدد 1234.560059 فهو تقريبا العدد 1234.56 الذي تم تعيينه للمتغير real_value. ونقول (تقريبا) لأن هناك فرقاً يساوى 000059. بين العددين .

لماذا هذا الخطأ؟

إنه ناتج من عملية تحويل العدد 1234.56 إلى النظام الثنائي حيث يصبح العدد كسرا غير منتهٍ ، ويضطر الحاسوب إلى قطعه حتى يسعه الحيز المخصص للجزء الكسري من الكلمة ، ولذلك فإن هذا الخطأ يعرف عادة باسم خطأ القطع أو خطأ التقريب roundoff error.

2.4 العمليات الحسابية

يوجد بلغة السي العمليات الحسابية الأربع وهى :

1. الجمع ، ويقوم بها المؤثر +
2. الطرح، ويقوم بها المؤثر −
3. الضرب، ويقوم بها المؤثر *
4. القسمة، ويقوم بها المؤثر /

إلى جانب عملية خامسة هي:

5. حساب باقي قسمة عددين صحيحين ، ويقوم بها المؤثر%.

لاحظ أن هذه الرموز الخمسة تسمى مؤثرات operators لأنها تقوم بالتأثير على قيمتين x , y (إما من النوع الصحيح أو الكسري) لتنتج قيمة جديدة . فمثلا في العملية :

$$z = x + y$$

يقوم المؤثر + بالتأثيرعلى القيمتين (x , y) ليعطى قيمة جديدة هي z.
وهذا ينطبق أيضا على باقي المؤثرات الحسابية، أي أنها تؤثر على قيمتين لتنتج قيمة ثالثة جديدة، لذلك فهي توصف بأنها مؤثرات ثنائية binary. ويمكن أن نستثني من ذلك المؤثر (–) فهوقد يؤثر أحيانا على قيمة واحدة لينتج معكوسها الضربي، وفي هذه الحالة يسمى مؤثرا أحاديا unary.

لاحظ عدم وجود مؤثر في لغة سي خاص بالأس، ولكن توجد دالة جاهزة تسمى

pow

تقوم بذلك، وسنقوم بدراستها مع موضوع الدوال الجاهزة.

بالنسبة إلى المؤثر الخاص بباقي قسمة عددين صحيحين وهو % فهو يعمل على النحو التالي:

$$14\%3 = 2$$

أي أن باقي قسمة 14 على 3 هو 2 . فإذا طلبنا من الحاسوب أن ينفذ الجملة التالية:

```
printf ( " %d", 23%5);
```

فإنه سيطبع العدد 3 وهو عبارة عن باقي قسمة 23 على 5.

من المهم عند تقييم عبـــارة حـــسابية arithmetic expression أن نعـــرف أسبقية المؤثرات ، أي ما هي العمليات التي ينفذها المصرّف سى قبل الأخرى ؟

أسبقية العمليات :

1 . العمليات المحصورة داخل القواسين ( ).
2 . عملية الضرب والقسمة والباقي .
3 . عملية الجمع والطرح .

في حالة تساوى الأسبقية (مثل الضرب والقسمة) يتم تنفيذ العملية الواقعة على اليسار قبل الأخـــرى .

مثال : العبارة الحسابية :

$$10 / 5 * (2 + 4)$$

تنفذ حسب الترتيب التالي:

$2 + 4 = 6$ (لأنها واقعة بين القوسين)

$10 / 5 = 2$

2 * 6 = 12

أي أن الناتج هو 12.

لاحظ أن عملية القسمة تمت قبل عملية الضرب لوقوعهاعلى اليسار.

<u>مثال</u> : العبارة الحسابية :

30 / 3 / 5

تنفد على النحو التالي :

30 / 3 = 10
10 / 5 = 2

أي أن الناتج هو 2 .

<u>مثال</u> : ما هو ناتج تنفيذ العبارة التالية ؟

2 + (5 * (6 + 7) + 3)

نلاحظ هنا أن الأقواس الداخلية يتم تنفيذها قبـــل الأقــواس الخارجيـــة، بمعنى أن أول ما يتم تنفيذه في هذه العبارة هو :

6 + 7 = 13

ثم

5 * 13 = 65

ثم

65 + 3 = 68

وأخيرا

2 + 68 = 70

<u>مثال</u> : ما هو ناتج القسمة 4 / 15 ؟

نلاحظ هنا أن 15 لا تقبل القسمة على 4 وبالتالي سيكون بالناتج كسر، ولكن المصرّف C يتعامل حسب القاعدة التالية:

ناتج قسمة عدد صحيح على عدد صحيح يساوى عددا صحيحا

بالتالي فإن الكسر يحذف من الناتج مما ينتج عنه أن :

15 /4 = 3

<u>مثال</u> : ما هو ناتج تنفيذ العبارة التالية ؟

3/6*(18+20)

في البداية يتم جمع 20 إلى 18 وينتج عن ذلك 38

ثم عملية القسمة :

3 / 6 = 0

وبالتالي فإن الناتج النهائي هو 0 حيث

0 * 38 = 0

مثال : ما هو ناتج العبارة الحسابية ؟

3. / 6 * (18 + 20)

الفرق الوحيد بين هذه العبارة والعبارة في المثال السابق هو وجود النقطة العشرية في .3 ولكنها تؤدى إلى نتيجة مختلفة تماما لأن

3. / 6 = .5

وذلك طبقا للقاعدة:

ناتج قسمة عدد عائم على عدد صحيح يعطى عددا عائما.

مثال : ما هو ناتج تنفيذ العبارة التـالية ؟

السؤال هنا هو:

هل يتم تنفيذ عملية الباقي % قبل أو بعد عملية الضرب * ؟
والجواب أن المؤثر (%) ينفذ قبل المؤثر (*) لوقوع الأول على يسار الثاني ولأنهما يتساويان في الأسبقية .
أي أن :

27 % 5 * 2 = 2 * 3 = 6

مثال : ما هو ناتج تنفيذ الآتي ؟

حيث أن المؤثر (%) له الأسبقية على المؤثر (-) فإن :

15 - 14 % 3 = 15 - 2 = 3

أي أن ناتج العملية هو 3.

<u>مثال</u> : ما هو ناتج تنفيذ الآتي :

15 + 14 % (- 3)

حيث أن ناتج قسمة 14 على 3 - هو 4- والباقي 2 لأن

14 / (- 3) = - 4 + 2 / (- 3)

فإن الباقي يساوى 2 . وبالتالي فإن:

15 + 14 % (- 3) = 17

2.5 التحويل من نوع إلى آخـــــر

ماذا يحدث إذا حددنا أن المتغير x من النوع العائم float وعيّنا لهذا المتغير قيمة صحيحة ؟

إن ما يحدث هنا هو عملية تحويل من الثابت الصحيح إلى الثابت العائم ثم تتعين القيمة العائمة للمتغير x .

<u>فمثلا</u> ينتج عن تنفيذ الجملة :

```
x = 8 ;
```

عـــملية تحويل العدد الصحيح 8 إلى العدد الكسري 8.0 لتتلاءم مع المتغير العائم x.

ولكن ماذا لو كان k من النوع الصحيح وعينا له ثابتا كسريا مثل؟ :

```
int   k ;
k = 3.6
```

في هذه الحالة لو طبعنا قيمة k لتحصلنا على 3 وليس 3.6 . أي أن الكسر يحذف كله ويبقى الجزء الصحيح فقط .

وبالتالي إذا تم تحديد المتغيرين x و k على النحو التالي :

```
int   k  ;
float  x  ;
```

فإن الجملة:

```
x = k ;
```

هي عملية تحويل من نوع صحيح إلى نوع عائم ، وهى تتم بدون فقدان كسور ، ولكن العملية :

```
k  =  x  ;
```

هي عملية تحويل من نوع عائم إلى صحيح ويحذف فيها الكسر ، ويتم تعيين الجزء الصحيح فقط من x إلى k .

هناك طريقة أخرى في لغة سى للتحويل من نوع إلى آخر . فإذا كان k من النوع الصحيح فإن :

(float) k

هي القيمة المقابلة له في النوع العائم . وإذا كان x من النوع العائم فإن :

(int) x

هي القيمة المقابلة له في النوع الصحيح ، حيث تساوى x محذوفا منها الجزء الكسري.

<u>مثال</u> : ماذا يطبع البرنامج التالي ؟

```
main()
{
        int k;
        float x;
        k=4;
        x=5.7;
        printf("\n %f", (float)k);
        printf("\n %d", (int)x);
}
```

الشكـــــل (2.5.1)

ناتج الطبـــاعــــــة هـــــــو :

4.000000
5

العدد الأول هو ناتج تحويل الثابت الصحيح 4 إلى النوع العائم .
والعدد الثاني هو ناتج تحويل الثابت العائم 5.7 إلى النوع الصحيح .

2.6 تمــــــــــــــارين

1 . اكتب برنامجا لطباعة العبارة " Good Morning " ابتداء من ســطر جديد على الشاشة .

2 . أي من المتغيرات التالية جائزة في لغة سى؟:

abc
ibm_pc
birth-date
if
SMALL
A^c
Sameer

3 . أوجد الأخطاء في البرنامج التالي :

```
main ( )
{
	float   a ; b ; c  ;
	a=0,5;
	b = 6.2
	c = a + b / 2  ;
			d =  3 * a  ;
	printf (( \ n % f  % f)  , a  , b ) ;
	printf (( \ n % d  % d ) , c , d )
}
```

4 . أعد كتابة البرنامج في تمرين (3) بصورة صحيحة ثم اكتب نـــاتج تنفيــذ البرنامج.

5 . هل يجوز كتابة جملة تعيين على النحو التالي:

k = k + 1;

6 . ماذا يحدث إذا نفذنا البرنامج التالي :

```
main ( )
{
        int big ;
        big = 333333 ;
        printf ( " \ n %d " , big ) ;
}
```

7 . اكتب العبارات الجبرية التالية بلغة سى:

a) $\frac{x + y}{a + b}$

b) $x + \frac{y}{z + w}$

c) $a - f\,\frac{b - a}{x - y}$

d) $\frac{a + b}{c(d + e)}$

$$2+5\left[6+\frac{x-y}{z+w}\right]$$ (e

8 . ما هو ناتج تنفيذ العمليات التالية بلغة السى:

a) 5 + 6 / 2
b) 5 + 6 / 7
c) (5 + 6) / 2
d) 16 % 3 * 5
e) 6 / 4 / 2
f) 2 + 15 / 3 * 2
g) 3 / 4 * 8
h) 3. / 4 * 8
i) 1.0 * 3 / 4 * 8
j) 1.0 e + 3 / 2.0 e + 2
k) 1.5 e -3 * 2 e + 4

9 . ما هي القيم التي تتعين للمتغيرات التالية :

```
float  x , y , z ;
int    i , j , k ;
i = 28 / 3   ;
j = 33 / 10 ;
k = 4 / 5 * 3 ;
x = 28 / 3 ;
y = 33 / 3 ;
z = 4. / 5 * 3 ;
```

10 . اكتب برنامجا لحساب التكلفة الإجمالية لبضاعة من ثلاثة أصناف على النحو التالي :

رقم الصنف	عدد القطع	تكلفة القطعة
1	15	36.527
2	27	14.123
3	105	8.125

11 . اكتب برنامجا لحساب عدد الساعات والدقائق والثواني في زمن قدره 10000ثانية.

الباب الثالث 3

الإدخال و الإخراج

Input & Output

3.1 تمهـــــيد

رغم أن المعلومات التي تم عرضها حتى الآن عن لغــة ســي تمكّــن الطالب من كتابة برنامج يقوم ببعض الحسابات وطباعتها ، إلا أن هذه البرامج من الناحية العملية غير مرضية . فهي أولا خاصة بإجراء حسابات على أعداد معينة ، كما أن إخراجها يتم بصورة غير واضحة.
نحاول في هذا الباب سد هذا النقص قبل أن ننطلق في الأبــواب القادمـــة إلـــى برامج أكثر فاعلية بإذن الله.

3.2 توضيح المخرجات

عند تنفيذ أي برنامج يجب أن تكون المخرجات واضحة المعنى، ويحبذ أن تكون باللغة التي يفهمها مشغّل البرنامج ، فإذا كان مشغّل البرنـــامج يفهـــم الانجليزية يمكنك أن تكتب ناتج البرنامج مثلا على الصورة:

total cost = 325.650 dinar

وهذا أفضل من أن يطبع العدد 325.650 بدون توضيح ماهيـــة هـــذا العـــدد. ولأداء هذا التوضيح ، يمكننا كتابة جملة printf على النحو :

```
printf( " \ n total cost = %f dinar " , cost ) ;
```

حيث سيطبع النضيد في هذه الجملة كما هو ، مع وضع القيمة الفعلية للمتغير cost مكان f% .

ولكن تبقى مشكلة واحدة لأن ناتج تنفيذ هذه الجملة سيكون على الصورة:

```
total cost =  325.650000  dinar
```

وذلك لأن الهيئة f% تعطي 6 خانات كسرية بينما المطلوب 3 خانات عشرية فقط.

ولحل هذه المشكلة يمكن إعادة كتابة الجملة على الصورة التالية :

```
printf ( " \ n total cost = % .3f  dinar " , cost )  ;
```

حيث أضفنا 3. وتعنى طباعة 3 خانات كسرية فقط .

يمكننا أيضا تحديد طول الحقل الذي تتم فيه طباعة العدد العائم وذلك على النحو:

```
printf ( " \ n total cost = % 7.3f  dinar " , cost ) ;
```

حيث طول الحقل المخصص لطباعة قيمة cost هو 7 . نلاحظ أهمية ترك فراغ بين 7.3f % وكلمة dinar كفاصل بينهما ، كما نلاحظ حساب النقطة العشرية ضمن خانات الحقل.

وإذا كان العدد المطبوع يمكن أن يكون سالبا فيجب حساب خانة الإشارة ضمن الحقل أيضا.

مثال (3.2.1) : ماذا يطبع البرنامج التالي عند تنفيذه؟

```
main()
{
        float price,t1,t2,total;
        int item, q;
        item=1;
        q=15;
        price=34.5;
        t1=q*price;
        printf("\n\n\tItem\tQty\tPrice\t\tCost");
        printf("\n\t%d\t%d\t%.3f\t\t%.3f",item, q,
                    price, t1);
        item=2;
        q=27;
        price=123.25;
        t2=q*price;
        printf("\n\t%d\t%d\t%.3f\t\t%.3f"
           ,item, q, price, t2);
        total=t1 + t2;
        printf("\n\n\t\t\t total cost=%7.3f dinar",total);
}
```

الشكل(3.2.1)

يطبع هذا البرنامج النتائج التالية:

Item	Qty	Price	Cost
1	15	34.500	517.500
2	27	123.250	3327.750

total cost = 3845.250 dinar

ونلاحظ في هذا البرنامج ما يلي:

1 . المتغيرات المستخدمة هي:

item رقم الصنف

quantity اختصار qty الكمية (عدد القطع)

price السعر

t1 و t2 التكلفة

total التكلفة الاجمالية

2 . استخدمنا (\t) في تنسيق الطباعة وهي تعني الانتقال إلى حقل جديد في نفس السطر. أي أن السطر يقسّم الى مجموعة من الحقول (الأعمدة) المتساوية الطول ، يمكننا باستخدامها أن نطبع النتائج على شكل جدول متناسق وواضح.

3 . لترك عمود (حقل) فارغا في الجدول نستخدم **\t** مــرتين متتاليتين ، أي **\t\t**.

3.3 توضيح وتوثيق البرنامج

كما نهتم بتوضيح المخرجات وجعلها مفهومة قــدر الإمكــان لمــشغل البرنامج ، يجب أن نهتم بوضوح البرنامج في حد ذاته أيضا . وذلك يعنى أنه :

1 . يجب أن تدل أسماء المتغيرات على مسمياتها. فبدلا من المتغير x كرمــز مثلا للعمر ، من الأفضل استخدام المتغير age الذي يعني العمر .

2 . يجب توضيح خطوات البرنامج بجمل تعليقيـــة (أي توضـــيحية)، وهــذه الجمل ليست موجهة للحاسوب بل للإنسان الذي يقــرأ البرنـــامج، وهـــى تساعده على فهمه وتتبعه .

في لغة C توضع الجمــل التوضــيحية (comment statements) بــين الرمزين :

/* ……… جمل توضيحية ………. */

وهى قد تكون باللغة الإنجليزية أو العربية أو بأي لغة أخرى.

مثال (3.3.1) : ماهو الغرض من البرنامج التالي؟:

```
 /* ****************************
 *            برنامج لحساب مساحة الدائرة
 ****************************** */
main()
{
       float radius = 5.67   ,
              /*  radius : نصف القطر */
           pi  = 22./7.   , /*  pi : مقدار ثابت  */
           area  ;         /*  area : المساحة */
       area = pi * radius * radius;
       printf( " \n  %f = المساحة " ,area );
}
```

الشكل (3.3.1)

لاحظ أن البرنامج بهذا الشكل واضح في متغيراته، والغرض منه هو حساب مساحة دائرة نصف قطرها 5.67 . لاحظ أيضا أنه بإمكانك وضع الجمل التوضيحية في أي مكان مناسب من البرنامج . ولاستخدام اللغة العربية في كتابة الجمل التوضيحية يجب تشغيل برنامج تعريب (مثل برنامج نافذة، أوالعمل تحت نظام تشغيل معرّب مثل Arabic DOS) قبل الدخول في turbo C.

هناك ملاحظة أخرى وهى حول الجملة :

area = pi * radius * radius ;

التي تكافئ الصيغة المعروفة :

المساحة = ط نق^2

حيث لم نستخدم رمزا خاصا للتربيع ولكن ضربنا نصف القطر radius في نفسه وذلك – كما ذكرنا سابقا – لعدم توفر رمز التربيع (أو الأس بصورة عامة) بلغة سى كما هو الحال في لغة فورتران أو لغة بيسك ، ولكن توجد الدالة pow التي تقوم بهذه الوظيفة والتي سوف نستخدمها فيما بعد .

3.4 إدخال البيانات

هناك ملاحظة أخرى لاشك أن القارئ قد انتبه إليها في البرنامج (الشكل 3.3.1) لحساب مساحة الدائرة، وهذه الملاحظة حول التحديد:

float radius = 5.67 ;

الذي يحدد أن المتغير radius من النوع العائم، وقيمته هي 5.67 .

إذا أردنا أن يكون البرنامج صالحا لحساب مساحة أي دائرة، نستخدم في لغة سى الدالة scanf لإدخال نصف القطر أو (أي بيانات أخرى) أثناء تنفيذ البرنامج . أي أن المصرّف C عند تنفيذه لهذه الدالة سيتوقف منتظرا إدخال قيمة المتغير المحدد في الدالة.

الصورة العامة لهذه الدالة هي :

```
scanf ( " string " , &var ) ;
```

حيث var هو رمز لأي متغير سواء أكان من النوع الصحيح أم العائم أم غير ذلك من الأنواع ، أما " string " فهو نضيد يوضح الهيئة التي يتم بها الإدخال تماما بنفس الطريقة التي اتبعناها في الدالة printf .

<u>فمثلا</u> الجملة: scanf (" % d " , &k) ;

تتطلب إدخال قيمة المتغير الصحيح k بدون تحديد طول الحقل . و إذا تطلب الأمر يمكننا تحديد طول الحقل ، وكمثال آخر ، الجملة :

```
scanf ( " % 5d " , &m ) ;
```

تتطلب إدخال قيمة للمتغير الصحيح m الذي يحتل حقلا واحدا متكونا من 5 خانات عشرية صحيحة .

إذا كان المتغير من النوع العائم يمكننا تحديد طول الحقل وأيضا تحديــد طــول الجزء الكسري. فمثلا الدالة :

```
scanf ( " % 10.3 f " , &x ) ;
```

تتطلب إدخال عدد كسري طول جزئه الكسري 3 خانات عشرية ، وطول العدد كله لا يزيد عن 10 خانات عشرية بما في ذلك النقطــة العــشرية والإشــارة والكسر.

بالإمكان أيضا إدخال قيمتين لمتغيرين في دالة واحدة .

فمثلا الجملة :

```
scanf ( " % f , % f " , &age , &w ) ;
```

تتــطلب إدخـــال قيمتـــين كسريـــتين. ويتم إدخال القيمتـــين في سطر واحـــد مثل :

25.5 , 75.6

مثال (3.4.1) : البرنامج التالي والمبين بالشكل (3.4.1) يحـــسب مـــساحة الدائرة بصورة عامة، أي أن نصف قطر الدائرة غير محدد بالبرنامج بـــل يـــتم إدخاله أثناء التنفيذ.

عند تنفيذ هذا البرنامج ، سيومض المؤشر على الشاشة في انتظار إدخال قيمة للمتغير radius ، ولن يقوم بالخطوة التالية إلا بعد إتمام ذلك وبالضغط على المفتاح enter.

```
/* ****************************
 *          برنامج لحساب مساحة الدائرة
 ****************************** */
main()
{
       float radius    , /*  radius : نصف القطر */
            pi  = 22./7.   , /*  pi : مقدار ثابت  */
            area  ;         /*  area : المساحة */
       scanf("%f",&radius);
       area = pi * radius * radius;
       printf( " \n  %f = المساحة " ,area );
}
```

الشكل(3.4.1)

في الحقيقة لازال هنـــاك نقص في هــذا البرنامج رغم تحسينه علـــى البرنامج بالشكل (3.3.1) ، وذلك لأن الذي يقوم بتشغيل هذا البرنامج لن يفهم المطلوب إدخاله إلا إذا كان على دراية بالغرض من البرنامج. كان من الأفضل أن يتم إظهار طلب على الشاشة على الصورة :

Please enter radius →

قبل استخدام الدالة scanf وذلك حتى يصبح واضحا أن الحاسوب في حالة انتظار إدخال قيمة نصف القطر من المشغّل . لذلك فإن تحسين البرنامج يصبح كالآتي:

```
 /* *****************************
  *           برنامج لحساب مساحة الدائرة
  ****************************** */
main()
{
     float radius   , /* radius : نصف القطر */
          pi = 22./7.   , /* pi : مقدار ثابت */
          area ;        /* area : المساحة */
     printf( " \n  Please eneter radius→  " );
     scanf( "  %f  " , &radius );
     area = pi * radius * radius;
     printf( " \n  %f = المساحة " ,area );
}
```

الشكل(3.4.2)

مرة أخرى نلاحظ إمكانية استخدام اللغة العربية إذا توفر لدينا برنامج تعريب مثل (nafitha) وبالتالي نكتب :

```
printf ( " \ n ادخل نصف القطر " ) ;
```

بدلا من

printf (" \ n enter radius ") ;

وأخيرا لابد أن القارئ يتساءل عن سبب وجود الرمز & فـي دالـة scanf ، والسبب لا يمكن توضيحه كاملا الآن إلا بعد دراسة الدوال وطرق تمرير القـيم إليها ، ولكن نكتفي هنا بالقول أن الرمز & يسمى بمـؤثر العنـوان address operator ، أي أنه مؤثر يدخل على متغير ليعطي عنوانه فـي الـذاكرة . وللمزيد من الاطلاع حول هذا المؤثر انظر موضوع المؤشرات والدوال .

البرنامج التالي يوضح كيف يمكن قراءة أكثر من متغير لدالة scanf واحدة :

```
/*  area of a triangle */
main()
{
  float h,w,a;
  printf("\n enter h,w->");
  scanf("%f,%f",&h,&w);
  a=w*h/2.;
  printf("\n area of triangle=%f",a);
}
```

الشكل (3.4.3) برنامج لحساب مساحة مثلث

إذا نفذنا هذا البرنامج ستظهر على الشاشة العبارة :

enter h , w →

في انتظار إدخال قيمة h (ارتفاع المثلث) وقيمة w (القاعدة) حيث يمكن إدخالهما في سطر واحد مع وضع الفاصلة بينهما مثل :

enter h , w → 2.3 , 4.5

وبعد الضغط على المفتاح enter يظهر الناتج على الصورة :

area of triangle = 5.175000

3.5 الملف stdio.h

هذا الملف هو أحد ملفات العناوين header files التي سوف تجد قائمة لها بالملحق في نهاية هذا الكتاب ، وبالإمكان الحصول عليها أيضا باستخدام المفتاح [F1] الذي يؤدى وظيفة المساعدة HELP .

يحتوى الملف stdio.h على تعاريف العديد من ثوابت وتراكيب الإدخال والإخراج ذات العلاقة بالدوال printf و scanf وغيرها . لذلك يستخدم التوجيه

include < stdio.h >

في بداية البرنامج الذي يستخدم أحد هذه الدوال . إلا أننا لم نكتب هذا التوجيه لغرض تبسيط البرنامج للمبرمج المبتدئ ، ولأنه يتم تلقائيا في المترجم

Turbo C ، ولكن من الأضمن في الحالة العادية وضعه في بداية البرنامج حتى يكون صالحا للتنفيذ على مختلف المترجمات .

نضيف الآن هذا التوجيه إلى البرنامج بالشكل (3.4.2) ليصبح علـــى النحـــو المبين بالشكل (3.5.1) .
دعنا الآن ندخل إلى قائمة المساعدة عـــن طريـــق المفتـــاح [F1] لنستكتـــشف معلومات أخرى عن الدالة scanf والدالة printf .
بتحـــريك المؤشـــر عن طريق الأســـهم حتـــى يـــصل إلـــى header files والضغط على المفتاح enter تظهر لـــنا قائـــمة ملفـــات العنـــاوين المميـــزة نحصل منها بنـــفس الطريقة على قائمـــة الـــدوال ذات العـــــلاقة بالإدخـــال والإخراج (input / output) .

```
/* area of a triangle */
#include <stdio.h>
main()
{
float h,w,a;
printf("\n enter h,w->");
scanf("%f,%f",&h,&w);
a=w*h/2.;
printf("\n area of triangle=%f",a);
}
```

الشكل (3.5.1)

دعنا أولا نحرك المؤشر إلى الدالة scanf لنحصل على نافذة معلومات عن هذه الدالة. نلاحظ من معلومات هذه النافذة أن الدالة scanf هي من النوع الصحيح، أي أنها ترجّع في اسمها قيمة عددية صحيحة هي عدد القيم المقروءة .

لنتأكد من ذلك ، دعنا نعيد برنامج حساب مساحة المثلث (كما في الشكل (3.5.2) مستفيدين من هذه المعلومات .

الآن دعنا ننفذ هذا البرنامج كالآتي :

enter h , w → 4.5 , 7.8

يظهر لنا الناتج على النحو التالي :

number of data items entered = 2
area of the triangle = 17.550001

```
#include <stdio.h>
main()
{
  int n;
  float h,w,a;
  printf("\n enter h,w->");
  n= scanf("%f,%f",&h,&w);
  a=w*h/2.;
  printf("\n
         number of data items entered=%d",n);
  printf("\n area of triangle=%f",a);
}
```

الشكل(3.5.2)

ملاحظة جانبية : هناك خطأ بسيط في حساب الحاسوب (!) حيث أن المساحة يجب أن تكون 17.55 فقط . من أين جاء هذا الخطأ ؟؟ ابحث عن الإجابة في كتب أساسيات الحاسوب.

بالنسبة للدالة printf يمكن الحصول على معلومات عنها بنفس لطريـــقة وذلك بواسطة عرض دوال الملف stdio.h في نافذة HELP .

لنلقي مثلا نظرة على البرنامج التالي:

```
/*  area of a triangle */
#include <stdio.h>
main()
{
  float h,w,a;
  int m,k;
  printf("\n enter h,w->");
  m=scanf("%f,%f",&h,&w);
  a=w*h/2.;
  printf("\n number of data items =%d",m);
  k=printf("\narea of triangle=%f",a);
  printf("\n number of bytes printed=%d",k);
}
```

الشكل (3.5.3)

نلاحظ أن الدالة printf، إلى جانب قيامها بطباعة القيم المطلوبة، تعطى في اسمها عدد الحروف والرموز number of bytes التي تمت طباعتها .
أي أن الجملة:

```
k = printf ( " \ n area  of  triangle = % f " , a ) ;
```

تقوم بعملين . الأول هو طباعة العبارة:

```
area  of  triangle = 6.000000
```

وذلك بافتراض أن المساحة = 6 ، و الثاني هو تعيين القيمة 26 للمتغير k ، حيث 26 هو عدد الرموز bytes (حروف وأرقام وفراغات) في هذه العبارة.

لذلك فإن تنفيذ البرنامج بالشكل (3.5.3) سيعطى النتائج التالية :

```
enter  h  ,  w  →    5  ,  4
number of data  items  =  2
area of triangle  = 10.000000
number of bytes printed = 27
```

3.6 تحديد الثوابت

في أغلب البرامج التطبيقية نجد استخداما لقيم ثابتة سواء أكانت قيما عددية أم رمزية ، مثل الثابت 3.14159 المستخدم في حساب مساحة الدائرة ، أو الثابت 2.54 المستخدم في التحويل من بوصة إلى سنتمتر ، ... الخ .

من الأفضل تحديد هذه الثوابت في بداية البرنامج ، وتسميتها بمسميات تدل عليها . فمثلا الثابت 3.14159 يمكن أن نطلق عليه الاسم المعروف به في الرياضيات وهو PI ، أما الثابت 2.54 المستخدم في التحويل من بوصة إلى سنتمتر فيمكن أن نطلق عليه الاسم INCH_TO_CM. لاحظ هنا استخدام الأحرف الكبيرة capital في أسماء الثوابت تمييزا لها عن المتغيرات التي سوف تستخدم لها الحروف الصغيرة ، وذلك تمشيا مع العرف المتبع في كتابة برامج بلغة سى .

و لتحديد الثوابت في لغة سي نستخدم التوجيه #define وذلك قبل الدالة main() كما في الشكل (3.6.1):

```
/*  area of a circle */
#include <stdio.h>
#define PI 3.14159
main()
{
       float r,a;
        printf("\n enter radius-->");
        scanf("%f",&r);
        a=PI*r*r;
        printf("\n area = %f",a);
}
```

الشكل (3.6.1) تعريف الثابت PI لحساب مساحة الدائرة

لاحظ أن التوجيه define# قد وضع في هذا البرنامج قبل الدالة main ( ) وبعد التوجيه include# ، ولكن هذا الترتيب غير ضروري طالما كانت كل التوجيهات التي تبدأ بالرمز # تأتى قبل الدالة main .

كمثال آخر للثوابت ، يبين البرنامج بالشكل (3.6.2) كيف يمكن الاستغناء عن أحد رموز لغة سى، وهو مؤثر العنوان &، واستبداله بمؤثر آخر أكثر وضوحا نختار له الاسم ADDRESS (أي عنوان) ، وذلك بواسطة التوجيه

#define ADDRESS &

```
/*  area of a circle مساحة الدائرة */

#include <stdio.h>
#define PI 22./7.
#define ADDRESS &
main()
{
      float r,a;
    printf("\n enter radius-->");
    scanf("%f",ADDRESS( r ) );
    a=PI*r*r;
    printf("\n area = %f",a);
}
```

الشكل (3.6.2) تعريف مؤثر العنوان

لاحظ أننا استخدمنا في هذا البرنامج جملة الإدخال:

```
scanf (" % f " , ADDRESS(r) ) ;
```

حيث وضعنا r بين قوسين كفاصل بين المؤثر ADDRESS (الذي حل محل &) والمتغير r ، وهو أمر ضروري وإلا سينتج خطأ رمز غير معروف undefined symbol إذا استخدمنا ADDRESS.

3.7 أمثلة متعددة

مثال (3.7.1) اكتب برنامجا لحساب إجمالي الدفع مقابل استهلاك الكهرباء بالدينار علما بأن تكلفة الكيلووات الواحد هي 200 درهم ، وأن هناك ضريبة قدرها 1.5% من قيمة الاستهلاك . اطبع الناتج على صورة :

please pay xxxx .xxx dinar

لاحظ أن الدينار = 1000 درهم .

في هذا المثال ، يمكننا استخدام المتغيرات التالية :

kilo = استهلاك الكهرباء بالكيلووات

v = قيمة الاستهلاك

الضريبة tax =

إجمالي الدفع pay =

```
main()
{       float kilo, v, tax, pay;
        printf("\n enter kilos-->");
        scanf("\n %f",&kilo);
        v=0.2*kilo;
        tax = 0.15 * v;
        pay = v  + tax;
        printf("\nPlease pay %8.3f dinar.",pay);
}
```

الشكل(3.7.1)

عند تنفيذ هذا البرنامج تظهر على الشاشة العبارة:

enter kilos →

طالبة إدخال قيمة الإستهلاك بالكيلووات. وبإدخال العدد 1234 مثلا يظهر الناتج على الشكل:

please pay 283.820 dinar .

يمكنك تحسين البرنامج بالشكل (3.7.1) بإضافة التوجيهات:

```
#include < stdio.h >
#define  DPK  0.2
#define  TR  0.15
```

حيث DPK اسم للمقدار الثابت 0.2 (تكلفة الكيلو الواحد) و TK للمقدار الثابت 0.15 (نسبة الضريبة) لأن هذه المقادير قد تتغير في يوم ما ومن الأفضل أن يجرى تعديلها خارج الدالة main مرة واحدة، من تتبع مواقعها داخل هذه الدالة وتعديلها.

<u>**مثال** (3.7.2)</u> اكتب برنامجا يقوم بقراءة الزمن بالثواني، ويحسب عدد الثواني والدقائق والساعات فى هذا الزمن .

قبل أن نكتب البرنامج المطلوب دعنا نحسب يدويا الثواني seconds والدقائق minutes والساعات hours في زمن sec قدره مثلا 10000 ثانية. الخطوة الأولى في هذا الحساب هي قسمة 10000 على 60 لنحصل على 166 دقيقة و الباقي 4 ثوان. ثم نقوم بقسمة 166 على 60 لنحصل على 2 ساعة والباقي 46 دقيقة. أي أن :

عدد الساعات=2

وعدد الدقائق=46

وعدد الثواني=4.

بصورة عامة فإن المعطيات هي :

الزمن بالثواني = sec

والمطلوب حساب :

عدد الساعات = hours

عدد الدقائق = minutes

عدد الثواني = seconds

إذا قسمنا sec على 60 فإن :

ناتج القسمة = الدقائق= min

باقي القسمة = الثواني = seconds

مرة أخرى نقوم بقسمة min على 60 لنحصل على :

ناتج القسمة = الساعات = hours

باقي القسمة = الدقائق = minutes

لاحظ أن المتغيرات يجب أن تكون كلها من النوع الصحيح حتى يكون ناتج القسمة صحيحا ، أما الباقي فيمكن حسابه باستخدام المؤثر % ، كما في الشكل (3.7.2).

```
/* Compute number of hours, minutes and seconds in time
seconds */
main()
{
```

```
        int sec , min , seconds, minutes, hours;
        printf ("\n time in seconds---> ");
        scanf("%d",&sec);
        min = sec/60 ;
        seconds = sec % 60 ;
        hours = min / 60 ;
        minutes = min % 60;
        printf("\n hours =%d ", hours);
        printf("\n minutes=%d ", minutes);
        printf("\n seconds=%d", seconds);
}
```

الشكل (3.7.2)

عند تنفيذ هذا البرنامج تظهر على الشاشة العبارة :

time in seconds →

أي أن المطلوب هو إدخال الزمن بالثواني. نفترض

الآن أن الادخال هو 12015 ، سيكون الناتج هو:

```
hours     =  3
minutes   = 20
seconds  = 12
```

ملاحظة : اعتمد البرنامج بالشكل (3.7.2) في حساب الساعات والـدقائق على مؤثر الباقي % الذي يحسب باقي قسمة عددين صحيحين . ماذا لو لم يكن هذا المؤثر متوفرا بلغة سى ؟ هل يمكننا حل هذه المسألة ؟

في الحقيقة يمكننا حساب الباقي بدون استخدام المؤثر % وذلك بمعرفة أن قسمة عددين صحيحين ينتج عنها عدد صحيح ، أي أن الكسر يهمل . فمثلا :

$$70 / 60 = 1$$

ولحساب الباقي في هذه الحالة يمكننا إجراء العملية التالية :

$$70 - (70 / 60) * 60 = 70 - 60 = 10$$

وذلك يعنى بصورة عامة أن

$$m - m / n * n = m \% n$$

وبالتالي يمكننا إعادة كتابة البرنامج بالشكل (3.7.2) باستخدام هذه العلاقـــة ، أي بدون استخدام المؤثر % .

3.8 تمارين

1 . ماذا يطبع البرنامج التالي ؟

```
main (  )
{
float  a , b ,c  ;
        a = 5.6  ;
        b = 3.4  ;
        c = 2.5  *  a / b  ;
       printf ( " \ n % f %f %f " , a , b , c )
}
```

2 . اكتشف الأخطاء في البرنامج التالي :

```
/ * wrong program * /
# include  stdio.h
{  int  i , j , k  ;
i = 5.1  ;
j = 6 / i  ;
k = PI  *  j  ;
printf (" wrong program " , i , j , k) ;
}
```

3 . اكتب برنامجا لحساب حجم ومساحة ال[illegible] علما بأن :

$$[illegible] = \frac{4}{3}\pi r^{3}$$

$$[illegible]^{2}$$

حيث v هو الحجم ، و A المساحة ، و r نصف القطر [illegible] مقدار ثابت يساوى تقريبا $\frac{22}{7}$.

لاحظ أن البرنامج يجب أن يقرأ قيمة نصف القطر .

وضع المخرجات على النحو التالي :

```
volume  of  sphere = ...................
   area  of  surface = ...................
```

4 . أعد كتابة البرنامج في تمرين (3) بحيث يتم الإخراج باستخدام هيئة النقطة العائمة .

5 . اكتب برنامجا لحساب تكلفة قطعة ارض مستطيلة الشكل وذلك بإدخال طول وعرض القطعة وتكلفة المتر المربع وطباعة الناتج على النحو التالي :

```
 width = xx . x  mater
 length = xx . x  meter
area = xxx .xx  square meter
```

cost / m = xxx . xxx dinar
total cost = xxxx.xxx dinar

6 . اكتب برنامجا لقراءة عددين صحيحين وطباعة :

1 . حاصل ضربهما

2 . حاصل القسمة وذلك على الصورة التالية :

(ناتج القسمة) quotient = xxx

(الباقي) remainder = xxx

7 . اكتب برنامجا لقراءة عدد صحيح (بالنظام العشري) وطباعة مقابله بالنظام الثماني والسادس عشري .

8 . اكتب برنامجا لحساب الزكاة المفروضة على مبلغ من المال علما بأن نسبة الزكاة هي ربع العشر .

9 . اشترى مواطن 3 أصناف من البضاعة وهى :

1 . الصنف الأول : وعدد القطع بهN1 وسعرالقطعة بالدينار P1

2 . الصنف الثاني : وعدد القطع به N2 وسعر القطعة P2

3 . الصنف الثالث : وعدد القطع به N3 وسعر القطعة P3

اكتب برنامجا لحساب التكلفة الإجمالية مع طباعة جميع المدخلات بشكل منسق .

10 . اكتب برنامجا لحساب عدد الثواني في زمن معلوم بعدد الساعات والدقائق والثواني.

11 . اكتب برنامجا لحساب عدد الأسابيع في عدد معلوم من الأيام مع كتابة الناتج على الصورة التالية :

xxxx days = xxx weeks + x days